AD-A198 467

REPORT DOCUMENTATION PAGE

1b. RESTRICTIVE MARKINGS	
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	3. DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
4. PERFORMING ORGANIZATION REPORT NUMBER(S) SOR-88-7	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Princeton University	6b. OFFICE SYMBOL (If applicable)
7a. NAME OF MONITORING ORGANIZATION U. S. Army Research Office	
6c. ADDRESS (City, State, and ZIP Code) Dept. of Civil Engineering/Operations Research Princeton, N.J. 08544	7b. ADDRESS (City, State, and ZIP Code) P. O. Box 12211 Research Triangle Park, NC 27709-2211
8a. NAME OF FUNDING/SPONSORING ORGANIZATION U. S. Army Research Office	8b. OFFICE SYMBOL (If applicable)
9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER	
8c. ADDRESS (City, State, and ZIP Code) P. O. Box 12211 Research Triangle Park, NC 27709-2211	10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.
11. TITLE (Include Security Classification) The Analysis of Averages and the Analysis of Variance	
12. PERSONAL AUTHOR(S) Colin Goodall	
13a. TYPE OF REPORT Technical	13b. TIME COVERED FROM ____ TO ____
14. DATE OF REPORT (Year, Month, Day) 1 April 1988	15. PAGE COUNT 30
16. SUPPLEMENTARY NOTATION The view, opinions and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other documentation.	
17. COSATI CODES: FIELD / GROUP / SUB-GROUP	18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

→ In the analysis of averages and analysis of variance of a balanced array, repeated sweeping out of margin summaries has several advantages. These include the use of resistant summaries as alternatives to the mean, an exploratory emphasis, and computational convenience. The advantages offset the computational inefficiency of the sweep method relative to the Yates algorithm. However, analysis of averages and variance is not straightforward using the sweep function commonly found in data analysis software. Part I describes an invariant data structure and enhancements to sweep to provide computation of degrees of freedom and organization and annotation of a sequence of sweeps. The usual analysis of variance and median polish are "defaults" of the enhanced functions. Part II and the Appendix detail the specific implementation in S. (LF) ←

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL Colin Goodall	22b. TELEPHONE (Include Area Code) 609-452-6487 / 22c. OFFICE SYMBOL

The Analysis of Averages and the Analysis of Variance

Part I. The Sweep Operator and Analysis of Variance
Part II. Implementation in the New S Environment
Appendix. Examples, manual pages, and S functions

Technical Report #SOR-88-7

1 April 1988

Colin Goodall

Program in Statistics and Operations Research
School of Engineering and Applied Science
Princeton University
Princeton, NJ 08544

ABSTRACT

In the analysis of averages and analysis of variance of a balanced array, repeated sweeping out of margin summaries has several advantages. These include the use of resistant summaries as alternatives to the mean, an exploratory emphasis, and computational convenience. The advantages offset the computational inefficiency of the sweep method relative to the Yates algorithm. However, analysis of averages and variance is not straightforward using the sweep function commonly found in data analysis software. Part I describes an invariant data structure and enhancements to sweep to provide computation of degrees of freedom and organization and annotation of a sequence of sweeps. The usual analysis of variance and median polish are "defaults" of the enhanced functions. Part II and the Appendix detail the specific implementation in S.

Part I. The Sweep Operator and Analysis of Variance

1. Introduction

Three computational approaches to the analysis of an additive model for a response with factors at 1, 2, 3 or more levels are 1. regression with dummy variables, 2. explicit computation of formulae for effects, 3. repetitive use of the sweep operator. In broad terms the least-squares regression approach (1) is best able to handle unbalanced tables, whereas an approach that utilizes the orthogonal structure (2) is computationally the fastest. For example, in SAS the GLM procedure handles the general linear model and the ANOVA procedure is optimized for balanced tables. The earliest algorithms for balanced data are those of Yates (1934, 1937). Modern generalizations of these algorithms (Schlater and Hemmerle, 1966, Chambers, 1977) are extensive and not trivial to code. The repetitive use of the sweep operator (3) is computationally the slowest of the three approaches, but has compensating advantages. These include:

a. Ease of generalization to statistical summaries other than the mean, as in median polish of 2-way tables (Tukey, 1977) and of 3-way tables (Cook, 1985). A careful algorithm for median polish (Velleman and Hoaglin, 1981) is very close to repetitive use of sweep. Other summaries include trimmed means (Emerson and Hoaglin, 1981, Becker and Chambers, 1984), the low-median (Siegel, 1983), and M-estimators of location.

b. An exploratory emphasis. An analysis of variance table is the conventional summary of the role of the various factors in a multi-way table. A more detailed decomposition is often desirable, as in graphical exploratory analysis of means and variance (Johnson and Tukey, 1987, Johnson, 1988). Prior to computing the ANOVA table, we compute statistical summaries of various margins of the table, paying special attention to location, scale, and outliers of the various batches of numbers involved. The repetitive use of sweep yields a sequence of tables of effects and residuals which are readily displayed and readily understood. Cobb (1984) gives examples of these tables, along with discussion and motivation for this algorithmic approach to ANOVA.

c. Computational convenience. A sweep operator is very easy to program and is standard in high-level software for data analysis, including APL, ISP (Donoho and Huber, 1986, Abrahams, 1985) and both old and new S (Becker and Chambers, 1984, Becker, Chambers, and Wilks 1988). If it is not explicitly included, the sweep operator is easily programmed using a function such as apply in S or reduce in ISP, which applies an arbitrary function to sections of an array. To analyze an additive model the arguments to sweep comprise the array, a 0, 1, 2 or more-way margin, and a statistical summary. Sweep returns the array *minus* the statistical summaries of the respective sections of the array. James and Wilkinson (1971) discuss the mathematical basis for use of the sweep operator.

2. Enhancements to Sweep

The sweep operator does not itself provide the ease of use, manageability, and completeness that we often desire. An enhanced sweep, called effect say, includes several additional features. The implementation of effect in new S is described in Part II.

1. *An invariant data structure.* The arguments to effect can be the result of a previous call to effect, plus any margin vector and statistical summary. Effect can be called repetitively with margins in any order. For example, in an analysis by means the main effects of the first factor of a table are zero when the first and second factor interaction effects have

already been removed.

2. *Degrees of freedom for analysis by means.* An array of residuals is constrained along one or more margins. These constraints, discussed in detail in Section 3, determine the degrees of freedom of the residuals. When the residual array is swept again, the constraints will possibly increase and the residual degrees of freedom possibly decrease. The difference between the two residual degrees of freedom is the degrees of freedom of the effects. In the above example (para. 1.) the corresponding main effects degrees of freedom are zero. The invariant data structure includes the constraints, which are updated by effect. For convenience the result of effect includes the degrees of freedom of the effects and of the residuals and argument arrays.

3. *Summary statistics for the effects, residuals and parent arrays.* In analysis by means the usual summary statistic is the sum of squares. For this and other analyses, alternative summaries may be desired. In general we are interested in a vector of summary statistics for each section of the array orthogonal to the chosen margin. The vector may include the sum of squared and absolute values, several measures of scale, the MAD and a standard deviation say, the number of outliers in the respective arrays, and other summaries. To ensure comparability between the summaries for the effects, residuals and parent array, the subarray of effects is duplicated to an array with the full dimensions (as in Cobb, 1984) before computing the summaries.

4. *Organized and annotated results.* The repeated use of effect produces multiple sets of results. A higher-level utility, called ANOVA say, collects together the effects, degrees of freedom, summary statistics and residuals from the repeated calls to effect. Each component of these four lists is labeled with the respective margins. A conventional anova table can then be obtained with very little additional effort.

5. *Conventional analyses are easy with anova.* Most additive analyses involve either a full analysis of variance, a main effects analysis, a main effects plus 1st-order interactions analysis, or several iterations of median polish. These are available as default options to anova.

Cobb (1984) describes an interactive ANOVA program with some of these features. It is written in BASIC for a 48K Apple II computer and therefore limited. The S environment (Becker and Chambers, 1984) includes the sweep operator and has some degree of flexibility, but not enough to easily support the enhancements. Both S and ISP include functions for the analysis of one- and two-way tables by means, medians, and some other robust summaries, but general tools for the analysis of multi-way tables are not widely available with either system. On the other hand, the New S environment (Becker, Chambers, and Wilks, 1988) has the powerful features that make programming the effect and anova functions practicable. Part II describes a New S implementation.

3. Residual Constraints

The invariant data structure returned by effect includes an array of residuals. An attribute of the residual array is a vector of constraints. The constraints, updated by effect, are used to compute the residual degrees of freedom. This section describes a specific formulation of the constraints attribute.

The constraints attribute is a logical vector of length 2^d, where d is the dimension of the array. Each element of the vector corresponds to one of the 2^d margins of the array, including the null margin (overall effect). A convenient association of the vector elements and margins uses the binary expansions of $0, 1, \ldots, 2^d - 1$. For example when $d = 5$, $19 = 10011$ (mod 2), and the 3-way margin containing the 1st, 2nd and 5th factors corresponds to the 20th element of the constraints vector.

A key observation is that the residuals array returned by effect is constrained not only on the margin that is the argument to effect but also on any subordinate margin. For example the margins subordinate to (1, 2, 5) are 0 (overall), 1, 2, 5, (1, 2), (1, 5), and (2, 5). The residuals array is constrained on the subordinate margins whether or not an overall analysis, main effects analysis, or any other call to effect preceded the most recent call. Therefore the constraints attribute is initialized FALSE and the elements corresponding to the margin and its subordinate margins are set TRUE with each successive call to effect. The indexes in the constraints vector of the subordinate margins are easily found by computing the binary expansion of $0, 1, \ldots, 2^{d'} - 1$, where d' is the order of the margin, then inserting 0's into these binary numbers corresponding to the factors not included in the margin, and converting back to decimal integers.

The degrees of freedom (df) associated with each margin is the sub-product of the respective elements of the dimension vector of the array, less 1. This is the df for the interaction when all lower order effects have been removed previously. The df of the residuals array is the sum of the df associated with each margin for which the constraint attribute is false. If $(e_1, \ldots, e_d)$ is the dimension vector then

$$\sum_{S \subset \{1, \ldots, d\}} \prod_{i \in S} (e_i - 1) = \prod_{i=1}^{d} e_i$$

The df of the effects is the difference between the df of the parent array and the df of the residuals. This difference of course depends on the previous analyses. For example the interaction effects for factor 1 and factor 2 of a $3 \times 8 \times 5$ table have 24 df if there are no previous analyses, 23 df if the overall mean of main effects for factor 3 is already removed, 16 df if the main effects for factor 1 or the (1, 3) interaction is already removed, *etc.*

Part II. Implementation in Sqpe

Part II sketches the implementation of repeated use of the sweep operator in New-S (Becker *et al,* 1987). Complete details are contained in the examples, S manual pages and the function listings in the Appendix.

1. Invariant Data Structure

The invariant data structure is an array, *arr*, a so-called *effects array*, with attributes *constraints* and *constraintsdf*. The logical *constraints* vector has length 2***dim(arr)*, and specifies the margins on which *arr* is constrained by previous sweeps, as described in Part I. The integer *constraintsdf* vector is included only for convenience. It is computed once from *dim(arr)* and contains the degrees of freedom associated with each margin. The elements of these two attribute vectors correspond to the margins in binary, not lexicographic order. This ordering is an advantage for updating (see the description of the *subordinates* function below), and is not a serious drawback because, knowing *dim(arr)*, the *constraintsdf* attribute indicates the corresponding margins.

Just as *dim(arr)* both gets and assigns the dimension attribute,

```
> dim(arr)
[1] 3 8 5
> dim(arr) <- c(3,8,5)
```

so we define analogous functions for the *constraints* attribute

```
"constraints" <- function(x) attr(x, "constraints")
"constraints<-" <- function(x, ct)
eval(substitute(attr(x, "constraints") <- ct), local = sys.parent(1))
```

and likewise for the *constraintsdf* attribute.

The function *is.effects* tests for the presence of these two attributes, the functions *as.effects* initializes the attributes. *as.effects* conditions its action on *is.effects*, to avoid overwriting an existing *constraints* vector.

```
"is.effects" <- function(x)
!is.null(attr(x, "constraints")) & !is.null(attr(x, "constraintsdf"))

"as.effects" <- function(x) {
if(!is.effects(x)) {
        revsubdim <- rev(dim(x) - 1)
        ncodes <- 2^length(revsubdim)
        prodrow <- substitute(function(y)prod(revsubdim[y]))
        attr(x, "constraints") <- rep(F, ncodes)
        attr(x, "constraintsdf") <- apply(mod(seq(0, ncodes - 1)), 1, prodrow) }
x
}
```

Note the use of *substitute* to ensure proper substitution of the local definition of *revsubdim* in the definition of the local function *prodrow*.

An example is

```
> dim(arr)
[1] 3 8 5
> is.effects(arr)
[1] F
> arr <- as.effects(arr)
> attributes(arr)
$dim:
[1] 3 8 5

$constraints:
[1] F F F F F F F F

$constraintsdf:
[1]  1  2  7 14  4  8 28 56
```

In the definition of *as.effects*, *mod* is an utility function that computes the expansion of an integer modulo an arbitrary base. The expansion of a single integer is a vector. An integer between 0 and *2**dim(arr) - 1*, expanded modulo the default base 2, specifies a margin of *arr*.

```
"mod" <-
function(numbers, base = 2, ncol = floor(log(max(numbers + 1e-05))/log(base))) {
if(any(numbers < 0)) stop("numbers must be non-negative")
mod <- base ** seq(ncol,0)
mode(mod) <- "integer"
res <- outer(numbers, mod, "%/%") %% base
if(base == 2) mode(res) <- "logical"
res
}
```

With the default *base=2* the conversion is binary, and the output is logical. The logical result includes the numeric properties of a matrix of 0's and 1's, so is a more flexible representation. The binary conversion of 0:7 is

```
> mod(0:7)
     [,1] [,2] [,3]
[1,]    F    F    F
[2,]    F    F    T
[3,]    F    T    F
[4,]    F    T    T
[5,]    T    F    F
[6,]    T    F    T
[7,]    T    T    F
[8,]    T    T    T
```

The use of *mod* base 10 converts integers into rows of digits.

```
> mod(seq(5,105,20),10)
     [,1] [,2] [,3]
[1,]    0    0    5
[2,]    0    2    5
[3,]    0    4    5
[4,]    0    6    5
[5,]    0    8    5
[6,]    1    0    5
```

mod has a twin function *modconc* that concatenates a vector to an integer, modulo the given base. We will use *modconc* later.

```
"modconc" <- function(mat, base = 2) {
if(any(mat < 0) | any(mat >= base)) stop(paste("digits must be between 0 and",
        base - 1, "inclusive"))
if(len(dim(mat)) == 1) mat <- matrix(mat, nrow = 1)
maxn <- ncol(mat) - 1
mod <- base ** (maxn:0)
as.integer(mat %*% mod)
}
```

When the same base is used, *modconc* is the inverse of *mod*

```
> modconc(mod(0:7))
[1] 0 1 2 3 4 5 6 7
```

but the two can also be used with different bases. For example

```
> modconc(a,10)
[1]   0   1  10  11 100 101 110 111
```

is the binary representation of 0:7.

2. Computation of effects

The *effect* function is central to the set. Its arguments include the invariant data structure, the margin over which to sweep, and the "average" function for sweeping with any of its additional arguments.

```
"effect" <- function(a, margin, fun, ...) {
if(is.list(a)) a <- a[["residuals"]]
if(!is.effects(a)) stop("Data is not an effects array; use as.effects()")
seqq <- seq(length(dim(a)))
if(is.logical(margin)) margin <- seqq[margin]
perm <- c(margin, seqq[ - margin])
effects <- apply(a, margin, fun, ...)
eff <- aperm(array(effects, dim(a)[perm]), order(perm))
res <- a - eff
list(effects = effects, residuals = res, arrayofeffects = eff)
}
```

The argument to *effect* can be an effects array or a list (the result of a previous call to *effect*) that contains a named element ***residuals*** that is itself an effects array. The effects array is tested for the presence of the *constraints* and *constraintsdf* attributes. A logical *margin* is converted to an integer margin. The heart of *effect* is a modified *sweep* function. The result is a list that includes the subarray of effects, the residuals array, and an array of effects with the same dimension of the argument array.

The residuals array must be an effects array. These is done by copying all the attributes of *a* to *res* and then modifying the *constraints* attribute by setting a subset of the attribute logical *TRUE*. (If that margin is already swept the *constraints* attribute does not change.) The degrees of freedom for the parent and residuals are then easily computed by summing the elements of *constraintsdf* for logical *FALSE* elements of *constraints*. The degrees of freedom for effects is the difference, and may be 0. The last lines of *effect* become

```
    ...    ...
attributes(res) <- attributes(a)
dimnames(eff) <- dimnames(a)
attr(res, "constraints")[subordinates(margin) + 1] <- T
df.parent <- sum(constraintsdf(a)[!constraints(a)])
df.residuals <- sum(constraintsdf(res)[!constraints(res)])
df.effects <- df.parent - df.residuals
df <- c(df.effects, df.residuals, df.parent)
list(effects = effects, df = df, residuals = res, arrayofeffects = eff)
}
```

The arguments of *effect* also include a summary statistic function, *ssfun*, by default sum-of-squares, which returns a vector of summary statistics for each of the effects, residuals, and parent arrays.

```
"effect" <- function(a, margin, ssfun = function(x)sum(x^2), fun = mean, ...) {
    ...    ...
ss <- drop(matrix(c(ssfun(eff), ssfun(res), ssfun(a)), ncol = 3))
list(effects = effects, df = df, ss = ss, residuals = res, arrayofeffects = eff)
}
```

Additional code in *effect* handles margin 0 (overall effect) and a character-vector margin. A character-vector margin is matched to *c("overall", names(dimnames(a)))*. Detailed documentation and the complete listing of *effect* is in the Appendix.

The ***subordinates*** function updates the *constraints* attribute. This function computes the positions in *constraints* corresponding to margins subordinate to the argument margin. This is equivalent to finding the integers that have binary expansions with 1's in a subset of the positions given in the argument, an integer vector. For example, 5, 1, 3 and 0 have binary expansions TFT, FFT, TFF and FFF. Each is subordinate to TFT, i.e. the argument vector c(1,3).

```
"subordinates" <- function(vec) {
if(vec == 0) return(0)
n <- 2^length(vec); vec <- rev(sort(vec)); nc <- vec[1]
mod2 <- mod(seq(0, n - 1))
mod2f <- matrix(F, n, nc)
mod2f[, nc + 1 - vec] <- mod2
modconc(mod2f)
}
```

3. Analysis of variance

A complete analysis of variance of the array *arr* requires *2**dim(arr)* calls to *effect*. A median polish of a 2-way or higher-way table involves a similar number of calls. *anova* is a high-level function to manage the results of these repeated calls to *effect*; *anova* concatenates the *effects* and *residuals* elements into lists, and the *df* and *ss* elements into a matrix and an array. The *names(dimnames(arr))* attribute of *arr* labels the factors of *arr* and is used to construct character-string names for each margin.

The arguments to *anova* include the array and a list of margins, in addition to the summary statistics function and average function with its optional arguments used in *effect*. A slightly simplified *anova* function is

```
"anova" <- function(a, margins, ssfun = function(x)sum(x^2), fun = mean, ...) {
m <- margins; n <- length(margins)
if(is.list(a)) a <- a[["residuals"]]
a <- as.effects(a)
effects <- df <- ss <- residuals <- rowlabels <- NULL
if(is.null(names(dimnames(a)))) namesvec <- seq(length(dim(a)))
      else {namesvec <- names(dimnames(a));names(namesvec)<-namesvec}
for(i in 1:n) {
      margin <- as.vector(m[[i]])
      a <- effect(a, margin, ssfun, fun, ...)
      effects <- c(effects, a["effects"])
      df <- rbind(df, a[["df"]])
      ss <- rbind(ss, a[["ss"]])
      residuals <- c(residuals, a["residuals"])
      if(all(margin == 0)) rowlabels <- c(rowlabels, "overall")
            else rowlabels <- c(rowlabels, paste(namesvec[margin], collapse = "*"))
}
collabels <- c("effects", "residuals", "parent")
dimnames(df) <- dimnames(ss) <- list(rowlabels, collabels)
names(effects) <- names(residuals) <- rowlabels
list(margins = m, df = df, ss = ss, effects = effects, residuals = residuals)
}
```

The *subsets* function is a utility function which is used to generate the margins. *subsets* computes all margins of the argument array, i.e. all subsets of the dimension vector of the array. An optional argument specifies the sizes of the subsets desired. The subsets are ordered lexicographically.

```
"subsets" <- function(a, sizes = NULL) {
nc <- length(dim(a)); seqq <- seq(0,nc); n <- 2 ** nc
mat <- mod(0:(n - 1))[, nc:1]
isizes <- apply(mat, 1, sum)
iperm <- order(isizes, 1:n)
if(!missing(sizes)) iperm <- iperm[!is.na(match(isizes[iperm], sizes))]
mat <- cbind(c(T, rep(F, n - 1))[iperm], mat[iperm,] )
ll <- NULL; for(i in 1:nrow(mat)) ll <- c(ll, list(seqq[mat[i, ]])); ll
}
```

The complete version of *subsets* (see Appendix) uses the *names(dimnames(a))* to construct a list of character vectors. An additional argument specifies whether to return a logical matrix instead of a list. Any one of these three forms is a valid argument to *anova*. Instead of the array, *subsets* may also take as argument an integer or character vector; it constructs all subsets of the specified sizes of the vector. The argument may also be a single integer, *nc* in the above definition, which becomes the vector argument *seq(0,nc)*. *subsets* is a general purpose function that is useful for all subsets regression and related analyses as well as with *anova*.

The final call to *anova* includes arguments *orders* (sizes) and *margins*. The default for *orders* is all orders but the highest, *length(dim(a))*. The default for *margins* is the logical-matrix result of *subsets* with arguments *a* and *orders*.

```
"anova" <- function(a, orders = 1:length(dim(a)) - 1,
        margins = subsets(as.array(a), sizes=orders, log=T),
        ssfun =function(x)sum(x^2), fun = mean, ...) { ...
```

4. ANOVA table and polishing

Conventional analyses of a multi-way table include an analysis of variance table. Also, when the same margin is swept repeatedly as in median polish, corresponding effects are summed. Two additional functions, *anovatable* and *polish*, modify the list returned by *anova* to include these features. Each function also eliminates all but the last array of residuals. If desired, both functions can be used, but only in the order *polish* then *anovatable*.

anovatable computes the conventional mean squares, F-ratios, and P-values. To shorten the result, the *df*, *ss* and *margins* elements are eliminated.

```
"anovatable" <- function(a) {
nr <- nrow(a$df)
df <- c(a$df[,1],a$df[nr,2]); ss <- c(a$ss[,1],a$ss[nr,2])
ms <- ss/df
fv <- ms/ms[nr+1]
pv <- 1-pf(fv,df,df[nr+1])
at <- cbind(df,ss,ms,fv,pv)
dimnames(at) <- list(c(dimnames(a$df)[[1]],"residual"),
      c("df","ss","ms","F-value","P-value"))
list(anovatable=at,effects=a$effects,
      residuals=a$residuals[[length(a$residuals)]])
}
```

The *polish* function uses a generalization of *tapply*. The *effects* element of the result of *anova* is a list of data vectors. *listapply* generalizes *tapply* to group a list of data vectors, rather than a vector of data.

```
"listapply" <- function(lis,ind,FUN="+") {
if(is.character(FUN)) FUN <- get(FUN)
seqq <- seq(length(ind))
dup <- duplicated(ind); whr <- seqq[!dup]
res <- list()
for(i in whr){
      inn <- ind[i]; tot <- 0
      for(j in seqq[ind==inn]) tot <- FUN(tot,lis[[j]])
      res <- c(res,list(tot))
      }
names(res) <- names(lis)[whr]
res
}
```

There is a further complication, namely the indices for grouping are themselves margins that make up either a list or a logical matrix. Since S does not support hashing list elements, a function *intlisttoint* converts a list of vectors of non-negative integers into an integer vector. *intlisttoint* converts each vector into the integer with binary expansion with 1's in the given positions *(cf. subordinates)*. For example c(1,3) expands to 5.

```
"intlisttoint" <- function(lis) {
        val <- integer(length(lis))
        for(i in seq(along=lis)){
                vec <- lis[[i]]
                if(vec == 0){val[i]_0;next}
                m <- rep(F,max(vec))
                m[vec] <- T
                val[i] <- as.integer(sum(m * 2**seq(0,length(m)-1)))
                }
        val
}
```

The *polish* function for a list of integer vector margins is

```
"polish" <- function(m) {
if(!is.list(m$margins)) stop("margins component must be a list")
list(df=m$df,ss=m$ss,effects=listapply(m$effects,intlisttoint(m$margins)),
        residuals=m$residuals[[length(m$residuals)]])
}
```

Acknowledgements

The computing used the Sqpe software made available to Princeton University for beta test. I wish to thank Rick Becker at AT&T Bell Laboratories for his assistance in using the new S, and Allan Wilks, also at Bell Laboratories, for an encouraging conversation. This research was supported in part in connection with research at Princeton University sponsored by the Army Research Office (Durham), DAALO3-86-K-0073.

References

Abrahams, D.M. (1985). "A tutorial introduction to Berkeley ISP," Documentation, Department of Statistics, University of California, Berkeley, CA.

Becker, R.A., and Chambers, J.M. (1984). *S, An Interactive Environment for Data Analysis and Graphics*, Belemont, CA: Wadsworth.

Becker, R.A., Chambers, J.M., and Wilks, A.R. (1988). *The New S Language: A Programming Environment for Data Analysis and Graphics*, Draft manual, Murray Hill, NJ: Bell Telephone laboratories.

Berkeley ISP (1986). Product announcement. Department of Statistics, University of California, Berkeley, CA.

Chambers, J.M. (1977). *Computational Methods for Data Analysis*, New York: John Wiley.

Cobb, G.W. (1984). "An algorithmic approach to elementary ANOVA," *American Statistician*, **38**, 120-123.

Cook, N. R. (1985). "Three-way analyses," in *Exploring Data Tables, Trends, and Shapes"* (D.C. Hoaglin, F. Mosteller, and J.W. Tukey, eds.), New York: John Wiley.

Emerson, J.D. and Hoaglin, D.C. (1983). "Analysis of two-way tables by medians," in *Understanding Robust and Exploratory Data Analysis* (D.C. Hoaglin, F. Mosteller, and J.W. Tukey, eds.), New York: John Wiley.

ISP/DGS/SGS (1985). *ISP User's Guide and ISP Command Descriptions*, Carlisle, MA: Artemis Systems.

Johnson, E.G. (1988). "Robust analysis of factorial designs via elemental subsets and outlier sterilization," Ph.D. Thesis, Department of Statistics, Princeton University.

Johnson, E.G. and Tukey, J.W. (1987). "Graphical exploratory analysis of variance illustrated on a splittering of the Johnson and Tsao Data," to appear in *Design, Data and Analysis by Some Friends of Cuthbert Daniel* (C.L. Mallows, ed.), New York: John Wiley.

PC-ISP (1986). *Interactive Scientific Processor*, (software package), Carlisle, MA: Artemis Systems.

Schlater, J.E. and Hemmerle, W.J. (1966). "Statistical computations based upon algebraically specified models," *CACM*, **9**, 865-869.

Siegel, A.F. (1983). "Low median and least absolute residual analysis of two-way tables," *Journal of the American Statistical Association*, **78**, 371-374.

Velleman, P.F. and Hoaglin, D.C. (1981). *Applications, Basics, and Computing of Exploratory Data Analysis*, Boston, MA: Duxbury.

Wilkinson, G.N. (1970). "A general recursive procedure for analysis of variance," *Biometrika*, **57**, 19-46.

Yates, F. (1934). "The analysis of multiple classification with unequal numbers in the subclasses," *JASA*, **29**, 51-66.

Yates, F. (1937). "The design and analysis of factorial experiments," *Imp. Bur. Soil. Sci. Tech. Comm.*, **35**.

Appendix

Example of effect

INFANT MORTALITY DATA   (Emerson and Hoaglin, 1983)

```
> infantmortality
               <=8 9-11   12 13-15 >=16
    Northeast 25.3 25.3 18.2  18.3 16.3
North Central 32.1 29.0 18.8  24.3 19.0
        South 38.8 31.0 19.3  15.7 16.8
         West 25.4 21.1 20.3  24.0 17.5
> names(dimnames(infantmortality))_c("Region","Education of Father")


> options(digits=2)
> effect(infantmortality,1)
Error in call to effect: Data is not an effects array; use as.effects()
> effect(as.effects(infantmortality),1)
$effects:
 Northeast North Central South West
        21            25    24   22

$df:
[1]  4 16 20

$ss:
[1] 10477   664 11141

$residuals:
               <=8  9-11   12 13-15 >=16
    Northeast  4.6  4.62 -2.5 -2.38 -4.4
North Central  7.5  4.36 -5.8 -0.34 -5.6
        South 14.5  6.68 -5.0 -8.62 -7.5
         West  3.7 -0.56 -1.4  2.34 -4.2
attr($residuals, "constraints"):
[1] T T F F
attr($residuals, "constraintsdf"):
[1]  1  3  4 12

$arrayofeffects:
              <=8 9-11 12 13-15 >=16
    Northeast  21   21 21    21   21
North Central  25   25 25    25   25
        South  24   24 24    24   24
         West  22   22 22    22   22
```

The main effects for region are computed without taking out the grand mean. Therefore there are 4 = 3+1 degrees of freedom for effects, and the usual 16 residual degrees of freedom.

Example of anova --- analysis by means

```
> a <- anova(infantmortality)
> names(a)
[1] "margins"   "df"        "ss"        "effects"   "residuals"
> a$margins
     [,1] [,2]
[1,]    F    F
[2,]    T    F
[3,]    F    T
> a$df
                    effects residuals parent
            overall       1        19     20
             Region       3        16     19
Education of Father       4        12     16
> a$ss
                    effects residuals parent
            overall   10420       721  11141
             Region      57       664    721
Education of Father     479       185    664
> a$effects
$overall:
 overall
      23

$Region:
 Northeast North Central South West
      -2.1           1.8   1.5 -1.2

$"Education of Father":
 <=8 9-11   12 13-15 >=16
 7.6  3.8 -3.7  -2.3 -5.4

> a$residuals[[length(a$residuals)]]
                <=8  9-11   12 13-15  >=16
    Northeast -2.95  0.84  1.2 -0.13  1.04
North Central -0.12  0.59 -2.2  1.91 -0.21
        South  6.91  2.91 -1.3 -6.37 -2.10
         West -3.84 -4.34  2.3  4.59  1.27
attr(, "constraints"):
[1] T T T F
attr(, "constraintsdf"):
[1]  1  3  4 12
```

The full analysis of variance includes computation of the overall effect, and effects for Region and Education of Father. Only the final table of residuals is printed here, instead of the 3 tables contained in a$residuals. The usual analysis of variance table is computed by anovatable.

```
> anovatable(a)$anovatable:
                    df    ss    ms F-value P-value
            overall  1 10420 10420   675.4  0.0000
             Region  3    57    19     1.2  0.3380
Education of Father  4   479   120     7.8  0.0025
           residual 12   185    15     1.0  0.5000
```

Example of anova --- analysis by medians

```
The arguments to anova include the function ssff, defined as follows:
> ssff
function(x)
{
        res <- as.single(c(mad(x)/qnorm(0.75), sqrt(var(c(x))), length(boxplot(
                x, plot = F)$out)))
        names(res) <- c("scaled mad", "std dev", "outliers")
        res
}
> mad
function(x)
median(abs(x - median(x)))


> m <- anova(infantmortality, margins = list(0, 1, 2, 1, 2),
                ssfun = ssff, fun = median)
> m
#    EDITED RESULTS
$df:
                    effects residuals parent
            overall       1        19     20
             Region       3        16     19
Education of Father       4        12     16
             Region       0        12     12
Education of Father       0        12     12

$ss:

, , scaled mad
                    effects residuals parent
            overall    0.00       5.6    5.6
             Region    2.08       5.3    5.6
Education of Father    4.52       1.6    5.3
             Region    0.82       1.4    1.6
Education of Father    0.00       1.4    1.4

, , std dev
                    effects residuals parent
            overall 2.2e-15       6.2    6.2
             Region 2.3e+00       6.2    6.2
Education of Father 4.1e+00       3.8    6.2
             Region 7.1e-01       3.6    3.8
Education of Father 2.1e-01       3.6    3.6

, , outliers
                    effects residuals parent
            overall       0         1      1
             Region       0         1      1
Education of Father       0         3      1
             Region       0         5      3
Education of Father       0         4      5

$effects:
$effects$overall:
 overall
      21

$effects$Region:
 Northeast North Central South West
      -2.4           3.6  -1.4  0.4

$effects$"Education of Father":
 <=8 9-11    12 13-15 >=16
 7.4  5.9 -0.45     0 -3.1
```

Example of anova --- analysis by medians

```
$effects$Region:
 Northeast North Central South  West
      0.35           -1.1  0.55 -0.55

$effects$"Education of Father":
 <=8 9-11    12 13-15 >=16
 0.4  0.4 -0.05   0.4    0


$residuals:
$residuals$overall:
               <=8 9-11   12 13-15 >=16
    Northeast  4.6  4.6 -2.5  -2.4 -4.4
North Central 11.4  8.3 -1.9   3.6 -1.7
        South 18.1 10.3 -1.4  -5.0 -3.9
         West  4.7  0.4 -0.4   3.3 -3.2

$residuals$Region:
               <=8 9-11   12 13-15 >=16
    Northeast  7.0  7.0 -0.1   0.0 -2.0
North Central  7.8  4.7 -5.5   0.0 -5.3
        South 19.5 11.7  0.0  -3.6 -2.5
         West  4.3  0.0 -0.8   2.9 -3.6

$residuals$"Education of Father":
               <=8 9-11    12 13-15  >=16
    Northeast -0.4  1.1  0.35   0.0  1.05
North Central  0.4 -1.1 -5.05   0.0 -2.25
        South 12.1  5.9  0.45  -3.6  0.55
         West -3.1 -5.9 -0.35   2.9 -0.55

$residuals$Region:
                 <=8 9-11   12 13-15 >=16
    Northeast -0.75  0.8  0.0 -0.35  0.7
North Central  1.55  0.0 -3.9  1.15 -1.1
        South 11.55  5.3 -0.1 -4.15  0.0
         West -2.55 -5.3  0.2  3.45  0.0

$residuals$"Education of Father":
               <=8 9-11    12 13-15 >=16
    Northeast -1.2  0.4  0.05 -0.75  0.7
North Central  1.2 -0.4 -3.85  0.75 -1.1
        South 11.2  4.9 -0.05 -4.55  0.0
         West -2.9 -5.7  0.25  3.05  0.0
```

The corresponding row and column effects are summed using polish.

```
> polish(m)$effects
$overall:
 overall
      21

$Region:
 Northeast North Central South  West
      -2.1            2.4 -0.85 -0.15

$"Education of Father":
 <=8 9-11   12 13-15 >=16
 7.8  6.2 -0.5   0.4 -3.1
```

anova	General ANOVA and analysis of averages for balanced arrays	anova

USAGE:

```
anova(a, orders=1:length(dim(a)) - 1,
  margins=subsets(as.array(a), orders, T),
  ssfun=function(x)sum(x^2), fun=mean, ...)
```

ARGUMENTS:

a:	An array to be analyzed by successive sweeps using effect().
orders:	The orders of the margins selected for computation of effects. The argument orders is matched to the rows of margins. The default is all orders. The orders argument is ignored if margins is not missing.
margins:	A list or matrix of margins selected for computation of effects. The default is all margins.
ssfun:	Summary statistic function, returning a vector of statistics for each successive effects, residuals and parent array -- see effect().
fun:	Function used to sweep out the margins.
	Optional additional arguments to fun. These are passed unchanged.

VALUE:

A named list of results.

margins:	The sequence of margins.
df:	A matrix of degrees of freedom for effects, residuals and parent arrays at each step. Each row of df is the df component of a single iteration of effect(). The row labels of the matrix are pasted subsets of names(dimnames(a)) or of 1:length(dim(a))
ss:	An array or matrix of summary statistics. If ssfun returns a vector with non-null names attribute then that names attribute is the 3rd element of dimnames(ss).
effects:	List of arrays of effects. Each element of the list is the effects component of a single iteration of effect().
residuals:	List of arrays of residuals.

EXAMPLES:

```
anova(a)   # full anova by means
anova(a,orders=c(0,1))   # main effects only
# 3-way median polish with alternative summary statistics:
"mad" <- function(x) median(abs(x - median(x)))
"ssff" <- function(x) {res <- as.real(c(mad(x)/qnorm(0.75), sqrt(var(c(x))),
                        length(boxplot(x,plot=F)$out)))
        names(res) <- c("scaled mad", "std dev", "outliers"); res }
anova(a,margins=list(0,1,2,3,1,2,3,1,2,3),ssfun=ssff,fun=median)
```

SEE ALSO:

effect
allsubsets

effect	Analysis of a multi-way array	**effect**

USAGE:

effect(a, margin, ssfun=function(x)sum(x^2), fun=mean. ...)

ARGUMENTS:

a: A numeric array, or a list containing a numeric array called residuals. The array should have attributes constraints and constraintsdf. These attributes are initialized by as.effects().

margin: The subscripts (integer, logical or character) over which the effects are computed. 0 is permissible and gives an overall effect. If margin is logical it must have length the number of dimensions of the array. If margin is a character vector margin it is matched to the names attribute of dimnames(a). margin="overall" matches dimension 0.

ssfun: Function to compute a vector of summary statistics. The default is sum-of-squares.

fun: Function used to compute the effects.

Optional additional arguments to fun. These are passed unchanged.

VALUE:

A named list of results.

effects: The computed effects. An array with dimension dim(a)[margin].

df: The degrees of freedom (df) of the effects, residuals, and parent arrays. These are determined by the constraints attribute (logical), the constraintsdf attribute (integer), and margin. The entries in constraintsdf are the partial products of dim(a)-1 (or dim(a$residual)-1), which are the df associated with each margin of the array. The parent df is the sum of the margin degrees of freedom corresponding to the TRUE elements of constraints(a) (or constraints(a$residual)). This constraints vector is updated by the margins that are subordinate to margin (see subordinates()). This is the constraints attribute of the residuals (below) and determines the residuals degrees of freedom. The effects degrees of freedom is the difference.

ss: Vector or matrix of summary statistics computed on the array of residuals or the array of effects. The three elements or columns of ss are the effects, residuals, and parent summaries.

residuals: Array of residuals with the same dimension as a or a$residuals. The residuals array contains the updated constraints attribute, and also inherits the subordinatesdim and dimnames attributes from a or a$residual.

arrayofeffects: Array of effects with the same dimension as a or a$residuals.

EXAMPLES:

```
> is.effects(a)
[1] F       # a has no constraints and constraintsdf attributes
> a <- as.effects(a)  # initialize the missing attributes
> a0 <- effect(a,0)   # take out grand mean
> a0.13 <- effect(a0,c(1,3))  # 2-way effects
> a0.13.1 <- effect(a0.13,1)  # this effects component is 0
> names(dimnames(a)) <- NULL
> effect(a,"anything")$df  # this is overall analysis by default
> names(dimnames(a))<-c("method","material","layer")
> effect(a,c("method","layer"))$df  # two-way analysis on method and layer
# one-way analysis by medians with logical margin:
> effect(a,c(T,F,F),fun=median)$df
```

SEE ALSO:

is.effects
as.effects
subordinates
anova

mod	Conversion of an integer to arbitrary base	mod

USAGE:

```
mod(numbers, base=2, ncol=floor(log(max(numbers+0.00001))/log(base)))
```

ARGUMENTS:

numbers:	A vector of positive integers to be converted.
base:	Base used in the conversion, default 2.
ncol:	Number of columns in the matrix of results.

VALUE:

A matrix with one row for each element of numbers. Each row contains positive integers in the range 0, 1, ... , base-1. The least-significant digit is on the right. If base=2 then the matrix is logical. A logical matrix is equivalent to a matrix of 0's (F) and 1's (T) for arithmetic operations.

EXAMPLES:

```
# binary conversion of 0:7
> a_mod(0:7)
    [,1] [,2] [,3]
[1,]   F    F    F
[2,]   F    F    T
[3,]   F    T    F
[4,]   F    T    T
[5,]   T    F    F
[6,]   T    F    T
[7,]   T    T    F
[8,]   T    T    T
# modconc() is the inverse of mod()
> modconc(a)
[1] 0 1 2 3 4 5 6 7
# modconc(a,base=10) gives the binary representation of 0:7.
> modconc(a,10)
[1]  0  1  10  11 100 101 110 111
# mod(.,base=10) converts integers into rows of digits
> mod(seq(5,105,20),10)
    [,1] [,2] [,3]
[1,]   0    0    5
[2,]   0    2    5
[3,]   0    4    5
[4,]   0    6    5
[5,]   0    8    5
[6,]   1    0    5
# modconc() is again the inverse of mod()
> modconc(mod(seq(5,105,20),10),10)
[1]  5 25 45 65 85 105
```

SEE ALSO:

modconc
odometer

modconc	Concatenate matrix of integers into a single column	**modconc**

USAGE:

modconc(mat, base=2)

ARGUMENTS:

mat: Matrix of integers each in the range 0, ..., base-1. Each row the matrix is represents a non-negative integer, modulo base, with the least-significant digit on the right. mat is typically the result of mod()

base: The base to be used in the concatenation, default 2.

VALUE:

A vector of integers. Each integer is the sum of the entries in the corresponding row of the matrix multiplied by base**(ncol(mat):1 - 1)

EXAMPLES:

See documentation for function mod()

SEE ALSO:

mod
odometer

subordinates	Binary-subordinate numbers	**subordinates**

USAGE:

subordinates(vec)

ARGUMENTS:

vec: A vector of non-negative integers

VALUE:

The vector of positive integers y with binary expansion subordinate to the vector x, where x is defined by x[vec]=T and x[!vec]=F. y is subordinate to x if !any(y[!x]), i.e. an element of y is true only if the same element of x is true.

EXAMPLES:

```
> subordinates(c(1,3))
[1] 0 1 4 5
```

```
# The binary expansions of 0, 1, 4 and 5 are FFF, FFT, TFF and TFT.
# Each is subordinate to TFT, which has T's in positions 1 and 3.
```

SEE ALSO:

effect
mod
modconc

subsets	compute all subsets	**subsets**

USAGE:

subsets(a, sizes=NULL, log=FALSE)

ARGUMENTS:

a:	A vector or array. If a is a single integer then subsets computes all subsets of 1:a. If a is a vector with length >= 2 then subsets computes all subsets of a. If a is an array then subsets computes all subsets of names(dimnames(a)) or of 1:length(dim(a)) if names(dimnames(a)) is null. Note: a vector is an array when its dim attribute is not null.
sizes:	The sizes of the subsets returned. Default all sizes, including 0.
log:	The subsets are returned as a logical matrix. Default FALSE.

VALUE:

Subsets in lexicographic order arranged in a list with numeric or character vectors for elements. When log=TRUE the value of subsets is a logical matrix. The logical matrix is the output of mod with rows sorted into lexicographic order and the order of columns reversed. The null, size 0, subset is 0 if a is a single integer or a is an array with names(dimnames(a)) null. The null subset is NA if a is a vector with length>=2, and is "overall" if a is an array with non-null names(dimnames(a)).

EXAMPLES:

```
> subsets(3) # all subsets of 1:3 including 0
> x <- 1:3
> subsets(x,log=T) # all subsets of 1:3 as a logical matrix
> y <- c("a","b","c")
> subsets(y,c(0,2)) # list of subsets of sizes 0 and 2
> subsets(anarray) # all subsets of dimensions of the array anarray
> subsets(as.array(x)) # the two trivial subsets of the array x
```

SEE ALSO:

anova
leaps

S functions for analysis of variance and averages

```
"anova" <-
function(a, orders = 1:length(dim(a)) - 1,
        margins = subsets(as.array(a), orders, T),
        ssfun =function(x)sum(x^2), fun = mean, ...)
{
#  This is a simple driver for the effect() function.
        m <- margins
        if(is.list(a)) a <- a[["residuals"]]
        a <- as.effects(a)
        effects <- df <- ss <- residuals <- rowlabels <- NULL
        collabels <- c("effects", "residuals", "parent")
        if(all(names(dimnames(a)) == "")) namesvec <- seq(length(dim(a)))
        else {namesvec <- names(dimnames(a));names(namesvec)<-namesvec}
        if(is.list(m)) {
                n <- length(m)
                getf <- substitute(function(i)
                as.vector(m[[i]]))
        }
        else {
                n <- nrow(m)
                getf <- substitute(function(i)
                as.vector(m[i,  ]))
        }
        for(i in 1:n) {
                margin <- getf(i)
                a <- effect(a, margin, ssfun, fun, ...)
                effects <- c(effects, a["effects"])
                df <- rbind(df, a[["df"]])
                ss <- rbind(ss, a[["ss"]])
                residuals <- c(residuals, a["residuals"])
                if(all(margin == 0)) rowlabels <- c(rowlabels, "overall")
                else rowlabels <- c(rowlabels, paste(namesvec[margin], collapse =
                        "*"))
        }
        dimnames(df) <- list(rowlabels, collabels)
        rratio <- nrow(ss)/length(rowlabels)
        if(rratio > 1) {
                ss <- aperm(array(ss, c(rratio, dim(df))),c(2,3,1))
                dimnames(ss) <- list(rowlabels,collabels,
                        names(ssfun(a[["residuals"]])))
        }
        else dimnames(ss) <- list(rowlabels, collabels)
        names(effects) <- names(residuals) <- rowlabels
        list(margins = m, df = df, ss = ss, effects = effects, residuals =
                residuals)
}
"as.effects" <-
function(x)
{
#  initialize the constraintcodes and constraintsubdim attributes
#  of an array.    There are 2^subdim codes.
        if(!is.effects(x)) {
                revsubdim <- rev(dim(x) - 1)
                ncodes <- 2^length(revsubdim)
                prodrow <- substitute(function(y)
                prod(revsubdim[y]))
                attr(x, "constraints") <- rep(F, ncodes)
                attr(x, "constraintsdf") <- apply(mod(seq(0, ncodes - 1)), 1, prodrow)
        }
        x
}
"constraints" <-
function(x)
attr(x, "constraints")
"constraints<-" <-
```

S functions for analysis of variance and averages

```
function(x, ct)
eval(substitute(attr(x, "constraints") <- ct), local = sys.parent(1))
"constraintsdf" <-
function(x)
attr(x, "constraintsdf")
"constraintsdf<-" <-
function(x, ct)
eval(substitute(attr(x, "constraintsdf") <- ct), local = sys.parent(1))
"effect" <-
function(a, margin, ssfun = function(x)
sum(x^2), fun = mean, ...)
{
#   compute a table of effects (with array of effect)
        if(is.list(a)) a <- a[["residuals"]]
        if(is.character(margin)){ margin <- match(margin, c("overall", names(
                dimnames(a)))) - 1
                margin <- margin[!is.na(margin)]}
        if(!is.effects(a)) stop(
                "Data is not an effects array; use as.effects()")
        if(is.character(fun)) fun <- get(fun)
        if(is.character(ssfun)) ssfun <- get(ssfun)
        if(all(margin == 0)) {
                effects <- fun(as.vector(a), ...)
                names(effects) <- "overall"
                eff <- array(effects, dim(a))
        }
        else {
                seqq <- seq(length(dim(a)))
                if(is.logical(margin)) margin <- seqq[margin]
                perm <- c(margin, seqq[ - margin])
                effects <- apply(a, margin, fun, ...)
                eff <- aperm(array(effects, dim(a)[perm]), order(perm))
        }
        res <- a - eff
        attributes(res) <- attributes(a)
        dimnames(eff) <- dimnames(a)
        attr(res, "constraints")[subordinates(margin) + 1] <- T
        df.oritotal <- prod(dim(a))
        df.parent <- df.oritotal - sum(constraintsdf(a)[constraints(a)])
        df.residuals <- df.oritotal - sum(constraintsdf(res)[constraints(res)])
        df.effects <- df.parent - df.residuals
        df <- c(df.effects, df.residuals, df.parent)
        ss <- drop(matrix(c(ssfun(eff), ssfun(res), ssfun(a)), ncol = 3))
        list(effects = effects, df = df, ss = ss, residuals = res, arrayofeffects
                 = eff)
}
"is.effects" <-
function(x)
!is.null(attr(x, "constraints")) & !is.null(attr(x, "constraintsdf"))
"mod" <-
function(numbers, base = 2, ncol = floor(log(max(numbers + 1e-05))/log(base)))
{
#    convert numbers to a matrix mod base
#    to reexpress as a single vector use
#    modconc (provided base <=10)
        if(any(numbers < 0)) stop("numbers must be non-negative")
        mod <- base ** (ncol:0)
        mode(mod) <- "integer"
        res <- outer(numbers, mod, "%/%") %% base
        if(base == 2) mode(res) <- "logical"
        res
}
"modconc" <-
function(mat, base = 2)
{
```

S functions for analysis of variance and averages

```
#    convert a matrix (or vector) to a vector (or scalar)
#    using base arithmetic
        if(any(mat < 0) | any(mat >= base)) stop(paste("digits must be between 0 and",
                base - 1, "inclusive"))
        if(len(dim(mat)) == 1) mat <- matrix(mat, nrow = 1)
        maxn <- ncol(mat) - 1
        mod <- base ** (maxn:0)
        as.integer(mat %*% mod)
}
"subordinates" <-
function(vec)
{
#   Find the numbers with binary expansions that are all dominated by
#   (or equal to) the binary number with T's in the positions given
#   in vec.   For example, the binary number with T's in the positions
#   1 and 3 (vec=c(1,3)) dominates or is equal to TFT, FFT, TFF, FFF
#   which expand to 5,1,3 and 0 respectively.
#
        if(vec == 0) return(0)
        ncodes <- 2^length(vec)
        vec <- rev(sort(vec))
        ncol <- vec[1]
        mod2 <- mod(seq(0, ncodes - 1))
        mod2full <- matrix(F, ncodes, ncol)
        mod2full[, ncol + 1 - vec] <- mod2
        modconc(mod2full)
}
"subsets" <-
function(a, sizes = NULL, log = FALSE)
{
#  Compute the complete list of margins for the vector or array.
        lnames <- NULL
        if(is.array(a)) {
                nc <- length(dim(a))
                if(!all(names(dimnames(a)) == "")) seqq <- c("overall", names(
                        dimnames(a)))
                else seqq <- 0:nc
        }
        else if(length(a) > 1) {
                nc <- length(a)
                seqq <- c(NA, a)
        }
        else {
                nc <- as.integer(a)
                seqq <- 0:nc
        }
        n <- 2 ** nc
        mat <- mod(0:(n - 1))[, nc:1]
        isizes <- apply(mat, 1, sum)
        iperm <- order(isizes, 1:n)
        if(!missing(sizes)) iperm <- iperm[!is.na(match(isizes[iperm], sizes))]
        mat <- mat[iperm,  ]
        if(log) return(mat)
        mat <- cbind(c(T, rep(F, n - 1))[iperm], mat)
        ll <- NULL
        for(i in 1:nrow(mat)) ll <- c(ll, list(seqq[mat[i,  ]]))
        ll
}
"apply" <-
function(X, margin, FUN, ...)
{
#  apply modified to return names attribute if a vector result and
#  dimnames not null; also handles margin=0 (returns names "overall").
        if(is.character(FUN)) FUN <- get(FUN)
        if(margin == 0) {
                ans <- FUN(as.vector(X), ...)
                if(!is.null(dimnames(X))) names(ans) <- "overall"
                return(ans)
        }
        d <- dim(X)
        dn <- dimnames(X)
        if(!is.null(dn)) dn <- dn[margin]
        ans <- NULL
        permvec <- c(seq(1, length(d))[ - margin], margin)
        newX <- aperm(X, c(seq(1, length(d))[ - margin], margin))
        subdim <- d[ - margin]
        newX <- matrix(newX, prod(subdim), prod(d[margin]))
        if(length(subdim) > 1) for(i in 1:ncol(newX)) ans <- c(ans, FUN(array(newX[,
                i], subdim), ...))
        else for(i in 1:ncol(newX)) ans <- c(ans, FUN(newX[, i], ...))
        if(length(margin) == 1 && length(ans) == ncol(newX)) {
                names(ans) <- dn[[1]]
                return(ans)
        }
        else if(length(ans) == ncol(newX)) return(array(ans, d[margin], dn))
        else if(length(ans) %% ncol(newX) == 0) {
                if(is.null(dn)) return(array(ans, c(length(ans)/ncol(newX),
                        d[margin])))
                else return(array(ans, c(length(ans)/ncol(newX), d[margin]),
                        c(list(NULL), dn)))
        }
        else return(ans)
}
"anovatable" <-
function(a) {
#  convert structure returned by anova() into an anovatable
nr <- nrow(a$df)
df <- c(a$df[,1],a$df[nr,2])
ss <- c(a$ss[,1],a$ss[nr,2])
ms <- ss/df
fv <- ms/ms[nr+1]
pv <- 1-pf(fv,df,df[nr+1])
at <- cbind(df,ss,ms,fv,pv)
dimnames(at) <- list(c(dimnames(a$df)[[1]],"residual"),
        c("df","ss","ms","F-value","P-value"))
list(anovatable=at,effects=a$effects,
        residuals=a$residuals[[length(a$residuals)]])
}
"listapply" <-
function(lis,ind,FUN="+"){
#  combine elements of list according to the indices ind
#  (hashintlist may be used to create an index vector ind from a list)
if(is.character(FUN)) FUN <- get(FUN)
seqq <- seq(length(ind))
dup <- duplicated(ind)
whr <- seqq[!dup]
newlis <- NULL
res <- list()
for(i in whr){
        inn <- ind[i]
        tot <- 0
        for(j in seqq[ind==inn]) tot <- FUN(tot,lis[[j]])
        res <- c(res,list(tot))
        }
names(res) <- names(lis)[whr]
res
}
"intlisttoint" <-
function(lis)
{
```

S functions for analysis of variance and averages

```
#   Hash a list of positive-integer vectors.
#   Expand the i'th object in the list into the integer with
#   binary expansion with 1's in the given positions
#   For example c(1,3) expands to 5.
#
        val <- integer(length(lis))
        for(i in seq(along=lis)){
                vec <- lis[[i]]
                if(vec == 0){val[i]_0;next}
                m <- rep(F,max(vec))
                m[vec] <- T
                val[i] <- as.integer(sum(m * 2**seq(0,length(m)-1)))
                }
        val
}
"polish" <-
function(m) {
# organize result of anova when polishing is used
if(!is.list(m$margins)) stop("margins component must be a list")
list(df=m$df,ss=m$ss,effects=listapply(m$effects,intlisttoint(m$margins)),
        residuals=m$residuals[[length(m$residuals)]])
}
```